Towards Using Promises for Multi-Agent Cooperation in Goal Reasoning

Daniel Swoboda, Till Hofmann, Tarik Viehmann, Gerhard Lakemeyer
Knowledge-Based Systems Group
RWTH Aachen University, Germany
{swoboda,hofmann,viehmann,lakemeyer}@kbsg.rwth-aachen.de

Abstract

Reasoning and planning for mobile robots is a challenging problem, as the world evolves over time and thus the robot's goals may change. One technique to tackle this problem is goal reasoning, where the agent not only reasons about its actions, but also about which goals to pursue. While goal reasoning for single agents has been researched extensively, distributed, multi-agent goal reasoning comes with additional challenges, especially in a distributed setting. In such a context, some form of coordination is necessary to allow for cooperative behavior. Previous goal reasoning approaches share the agent's world model with the other agents, which already enables basic cooperation. However, the agent's goals, and thus its intentions, are typically not shared.

In this paper, we present a method to tackle this limitation. Extending an existing goal reasoning framework, we propose enabling cooperative behavior between multiple agents through promises, where an agent may promise that certain facts will be true at some point in the future. Sharing these promises allows other agents to not only consider the current state of the world, but also the intentions of other agents when deciding on which goal to pursue next. We describe how promises can be incorporated into the goal life cycle, a commonly used goal refinement mechanism. We then show how promises can be used when planning for a particular goal by connecting them to timed initial literals (TILs) from PDDL planning. Finally, we evaluate our prototypical implementation in a simplified logistics scenario.

1 Introduction

While classical planning focuses on determining a plan that accomplishes a fixed goal, a goal reasoning agent reasons about which goals to pursue and continuously refines its goals during execution. It may decide to suspend a goal, re-prioritize its goals, or abandon a goal completely. Goals are represented explicitly, including goal constraints, relations between multiple goals, and tasks for achieving a goal. Goal reasoning is particularly interesting on mobile robots, as a mobile robot necessarily acts in a dynamic environment and thus needs to be able to react to unforeseen changes. Computing a single plan to accomplish an overall objective is often unrealistic, as the plan becomes unrealizable during execution. While planning techniques such as contingent planning (Hoffmann and Brafman 2005), conformant planning (Hoffmann and Brafman 2006), hierarchical planning (Kaelbling and Lozano-Pérez 2011), and continual planning (Brenner and Nebel 2009; Hofmann et al. 2016) allow to deal with some types of execution uncertainty, planning for the overall objective often simply is too time consuming. Goal reasoning solves these problems by splitting the objective into smaller goals that can be easily planned for, and allows to react to changes dynamically by refining the agent's goals. This becomes even more relevant in multi-agent settings, as not only the environment is dynamic, but the other agents may also act in a way that affects the agent's goals. While some work on multi-agent goal reasoning exists (Roberts et al. 2015, 2021; Hofmann et al. 2021), previous approaches focus on conflict avoidance, rather than facilitating active cooperation between multiple agents.

In this paper, we propose an extension to multi-agent goal reasoning that allows effective collaboration between agents. To do this, we attach a set of *promises* to a goal, which intuitively describe a set of facts that will become true after the goal has been achieved. In order to make use of promises, we extend the goal lifecycle (Roberts et al. 2014) with *goal operators*, which, similar to action operators in planning, provide a formal framework that defines when a goal can be formulated and which objective it pursues. We use promises to evaluate the goal precondition not only in the current state, but also in a time window of future states (with some fixed time horizon). This allows the goal reasoning agent to formulate goals that are currently not yet achievable, but will be in the future. To expand the goal into a plan, we then translate promises into *timed initial literals* from PDDL2.2 (Edelkamp and Hoffmann 2004), thereby allowing a PDDL planner to make use of the promised facts to expand the goal. With this mechanism, promises enable active collaborative behavior between multiple, distributed goal reasoning agents.

The paper is organized as follows: In Section 2, we summarize goal reasoning and discuss related work. As basis of our implementation, we summarize the main concepts of the CLIPS Executive (CX) in Section 3. In Section 4, we introduce a formal notion of a goal operator, which we will use in Section 5 to define promises. In Section 6, we evaluate our approach in a distributed multi-agent scenario, before we conclude in Section 7.

2 Background and Related Work

Goal Reasoning. In goal reasoning, agents can "deliberate on and self-select their objectives" (Aha 2018). Several kinds of goal reasoning have been proposed (Vattam et al. 2013): The *goal-driven autonomy* (GDA) framework (Muñoz-Avila et al. 2010; Cox 2016) is based on finite state-transition systems known from planning (Ghallab, Nau, and Traverso 2016) and define a goal as conjunction of first-order literals, similar to goals in classical planning. They define a goal transformation function β, which, given the current state s and goal g, formulates a new goal g'. In the GDA framework, the goal reasoner also produces a set of *expectations*, which are constraints that are predicted to be fulfilled during the execution of a plan associated with a goal. In contrast to promises, which we also intend as a model of constraints to be fulfilled, those expectations are used for discrepancy detection rather than multi-agent coordination. The *Teleo-Reactive Executive* (T-REX) architecture (McGann et al. 2008) is a goal-oriented system that employs multiple levels of reasoning abstracted in reactors, each of which operates in its own functional and temporal scope (from the entire mission duration to second-level operations). Reactors on lower levels manage the execution of subgoals generated at higher levels, working in synchronised timelines which capture the evolution of state-variables over time. While through promises we also attempt to encode a timeline of (partial) objective achievement, our approach has no hierarchy of timelines and executives. Timing is not used to coordinated between levels of abstraction on one agent, but instead is used to indicate future world-states to other, independent agents reasoning in the same temporal scope. A *Hierarchical Goal Network* (HGN) (Shivashankar et al. 2012) is a partial order of goals, where a goal is selected for execution if it has no predecessor. HGNs are used in the GoDeL planning system (Shivashankar et al. 2013) to decompose a planning problem, similar to HTN planning. Roberts et al. (2021) extend HGNs to *Goal Lifecycle Network* (GLNs) to integrate them with the goal lifecycle. The goal lifecycle (Roberts et al. 2014) models goal reasoning as an iterative refinement process and describes how a goal progresses over time. As shown in Figure 1, a goal is first *formulated*, merely stating that it may be relevant. Next, the agent *selects* a goal that it deems the goal to be useful. The goal is then *expanded*, e.g., by querying a PDDL planner for a plan that accomplishes the goal. A goal may also be expanded into multiple plans, the agent then *commits* to one particular plan. Finally, it *dispatches* a goal by starting to execute the plan. It has been implemented in *ActorSim* (Roberts et al. 2016) and in the CX (Niemueller, Hofmann, and Lakemeyer 2019), which we extend in this work with promises. Most goal reasoning approaches focus on scenarios with single agents or scenarios where a central coordinating instance is available. In contrast, our approach focuses on a distinctly decentralized multi-agent scenario with no central coordinator.

Multi-Agent Coordination. In scenarios where multiple agents interact in the same environment, two or more agents might require access to the same limited resources at the same time. In such cases, multi-agent coordination is necessary to avoid potential conflicts, create robust solutions and enable cooperative behavior between the agents (Dias et al. 2006). Agents may coordinate implicitly by recognizing or learning a model of another agent's behavior (Sukthankar et al. 2014; Albrecht and Stone 2018). Particularly relevant is *plan recognition* (Carberry 2001), where the agent tries to recognize another agent's goals and actions. ALLIANCE (Parker 1998) also uses an implicit coordination method based on *impatience* and *acquiescence*, where an agent eventually takes over a goal if no other agent has accomplished the goal (impatience), and eventually abandons a goal if it realizes that it is making little progress (acquiescence). In contrast to such approaches, we propose that the agents explicitly communicate (parts of) their goals, so other agents may directly reason about them. To obtain a conflict-free plan, each agent may start with its own individual plan and iteratively resolve any flaws (Cox and Durfee 2005), which may also be modeled as a distributed constraint optimization problem (Cox, Durfee, and Bartold 2005). Instead of starting with individual plans, Jonsson and Rovatsos (2011) propose to start with some initial (sub-optimal) shared plan and then let each agent improve its own actions. Alternatively, agents may also *negotiate* their goals (Davis and Smith 1983; Kraus 2001), e.g., using *argumentation* (Kraus, Sycara, and Evenchik 1998), which typically requires an explicit model of the mental state. Verma and Ranga (2021) classify coordination mechanisms based on properties such as static vs dynamic, weak vs strong, implicit vs explicit, and centralized vs decentralized. Here, we focus on dynamic, decentralized and distributed multi-robot coordination. Alternatively, coordination approaches can be classified based on organization structures (Horling and Lesser 2004), e.g., coalitions, where agents cooperate but each agent attempts to maximize its own utility, or teams, where the agents work towards a common goal. Such teams may also form dynamically, e.g., by dialogues (Dignum, Dunin-Keplicz, and Verbrugge 2001). Here, we are mainly interested in fixed teams, where a fixed group of robots have a common goal. Role assignment approaches attempt to assign fixed and distinct roles to each of the participating agents and thereby fixing the agent's plan. This can be achieved either by relying on a central instance or through distributed approaches (Iocchi et al. 2003; Vail and Veloso 2003; Jin et al. 2019). Intention sharing approaches allow agents to incorporate the objectives of other agents into their own reasoning in an effort to preemptively avoid conflicting actions (Holvoet and Valckenaers 2006; Grant, Kraus, and Perlis 2002; Sarratt and Jhala 2016). *SharedPlans* (Grosz and Kraus 1996) use complex actions that involve multiple agents. They use an intention sharing mechanism, formalize collaborative plans, and explicitly model the agents' mental states, forming mutual beliefs. Similar to promises, an agent may share its intention with *intending-that* messages, which allow an agent to reason about another agent's actions. Depending on the context, agents must have some form of *trust* (Yu et al. 2013; Pinyol and Sabater-Mir 2013) before they cooperate with other agents. Market-based approaches as surveyed by Dias et al. (2006) use a bidding process to allocate different conflict-free tasks between the agents of a multi-robot system. This can be extended to auction of planning problems in an effort to optimize temporal multi-agent planning (Hertle and Nebel 2018). In the context of multi-agent goal reasoning, Wilson and Aha (2021) perform the goal lifecycle for an entire multi-robot system on each agent separately and then use optimization methods to prune the results.

3 The CLIPS Executive

The CLIPS Executive (CX) (Niemueller, Hofmann, and Lakemeyer 2019) is a goal reasoning system implemented in the CLIPS rule-based production system (Wygant 1989). Goals follow the goal lifecycle (Roberts et al. 2014), with domain specific rules guiding their progress. It uses PDDL to define a domain model, which is parsed into the CLIPS environment to enable continuous reasoning on the state of the world, using a representation that is compatible with a planner. Therefore, the notions of plans and actions are the ones known from planning with PDDL. Multiple instances of the CX (e.g., one per robot), can be coordinated through locking and domain sharing mechanisms, as demonstrated in the RoboCup Logistics League (RCLL) (Hofmann et al. 2021). However, these mechansims only serve to avoid conflicts as each agent still reasons independently, without considering the other agents intents and goals.

Goals. Goals are the basic data structure which model certain objectives to be achieved. Each goal comes with a set of preconditions which must be satisfied by the current world state before being formulated. In the CX, these preconditions are modelled through domain-specific CLIPS rules.

Each instance of a goal belongs to a class, which represents a certain objective to be achieved by the agent. Additionally, the goal can hold parameters that might be required for planning and execution. A concrete *goal* is obtained by grounding all parameters of a goal class. Depending on the current state of the world, each robot may formulate multiple goals and multiple instances of the same goal class.

Each goal follows a *goal lifecycle*, as shown in Figure 1. The agent formulates a set of goals based on the current state of the world. These can be of different classes, or multiple differently grounded versions of the same class. After formulation, one specific goal is selected, e.g., by prioritizing goals that accomplish important objectives. The selection mechanism may be modelled in a domain-specific way, which allows adapting the behavior according to the domain-specific requirements. In the next step, the selected goal is *expanded* by computing a plan with an off-the-shelf PDDL planner. Alternatively, a precomputed plan can be fetched from a plan library. Once a plan for a goal is determined, the executive attempts to acquire its required resources. Finally, the plan is executed on a per-action basis.

Goals Example. To illustrate goals and goal reasoning in the CX, let us consider a simple resource mining scenario which we will evolve throughout this paper and finally use as the basis for our conceptual evaluation. Different case examples based on the given setting are presented in Figure 2. Up to two robots (WALL-E and R2D2) produce the material *Xenonite* by first mining *Regolith* in a mine, refining the Regolith to *Processite* in a refinery machine, and then using Processite to produce Xenonite in a production machine. Let us assume the following setting as the basis for our example: one machine is filled with Regolith, one machine is empty, the robots currently pursue no goals. In this scenario, each robot may pursue five classes of goals: (a) FILLCONTAINER to fill a container with Regolith at the mine or with Processite, (b) CLEANMACHINE to fill a container with Processite or Xenonite at a machine, (c) DELIVER a container to a machine, (d) STARTMACHINE to use a refinery or production machine to process the material, (e) DELIVERXENONITE to deliver the finished Xenonite to the storage. Because one machine is filled, a goal of class STARTMACHINE can be formulated. The parameters of the goal class are grounded accordingly, i.e., the target machine is set to the filled machine. Should both machines be filled at the same time, two goals of class STARTMACHINE could be formulated, one for each of the machines. The agent now selects one of the formulated goals for expansion. Since in our case only one goal is formulated, it will be selected. Should there be START-MACHINE goals for each machine, then, e.g., the one operating on second machine in the production chain might be prioritized.

Execution Model. After a goal has been expanded (i.e., a plan has been generated), the CX commits to a certain plan and then dispatches the goal by executing the plan. For sequential plans, whenever no action of the plan is executed, the CX selects the next action of the plan. It then checks whether the action is executable by evaluating the action's precondition, and if so, sends the action to a lower level execution engine. If the action's precondition is not satisfied, it remains in the pending state until either the precondition is satisfied or a timeout occurs. In a multi-agent scenario, each CX instance performs its own planning and plan execution independently of the other agents.

The execution of two goals with one robot in our example scenario is visualized in with one robot is visualized in Figure 2 as SCENARIO 1. After finishing the goal of class STARTMACHINE, a goal of class CLEANMACHINE is formulated and dispatched, clearing the machine's output.

Multi-Agent Coordination. The CX supports two methods for multi-agent coordination: *locking actions* acquire a lock for exclusive access, e.g., a location so no two robots try to drive to the same location. They are part of a plan and executed similar to regular actions. *Goal resources* allow coordination on the goal rather than the action level. Each goal may have a set of *required resources* that it needs to hold exclusively while the goal is dispatched. If a second goal requires a resource that is already acquired by another goal, the goal is rejected. This allows for coordination of conflicting goals, e.g., two robots picking up the same container. Both methods are intended to avoid conflicts and overlapping goals between the agents, rather than fostering active cooperation.

To illustrate coordination by means of goal resources, let us extend the previous example SCENARIO 1 by adding the second robot. In SCENARIO 2, illustrated in Figure 2, both WALL-E and R2D2 formulate and select STARTMACHINE for `M1`, which requires the machine as a resource. WALL-E starts formulating first and manages to acquire the resources. Therefore it is able to dispatch its goal. R2D2 on the other hand must reject the goal as it is not able to acquire the machine resource. After its goal has been rejected, it may select a different goal. Since now the machine is occupied, it has to wait until the preconditions of some class are met and a new goal can be formulated. This is the case once the effects of STARTMACHINE are applied, which causes R2D2 to formulate and dispatch CLEANMACHINE to clear the machine.

Execution Monitoring. The CX includes execution monitoring functionality to handle exogenous events and action failure (Niemueller, Hofmann, and Lakemeyer 2018). Since most actions interact with other agents or the physical world, it is critical to monitor their execution for failure or delays. Action may either be retried or failed, depending on domain-specific evaluations of the execution behavior. Additionally, timeouts are used to detect potentially stuck actions. The failure of an action might lead to reformulation of a goal. This means that the plan of the goal might be adapted w.r.t. the partial change in the world state since original formulation. Alternatively, a completely new objective might be followed by the agent.

4 Goal Reasoning with Goal Operators

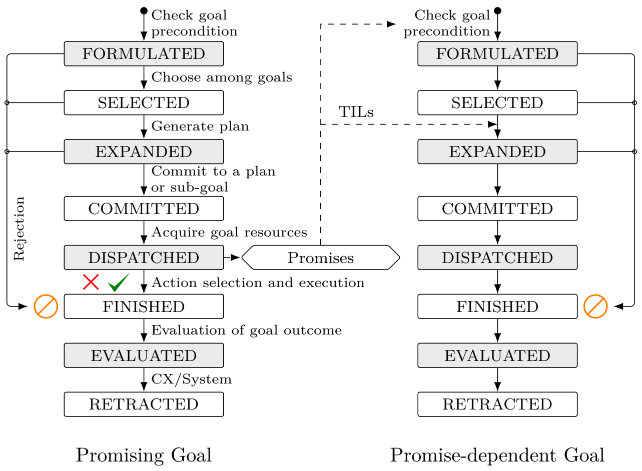


Figure 1: The CX goal lifecycle (Niemueller, Hofmann, and Lakemeyer 2019) of two goals and their interaction by means of promises. When the first goal is dispatched, it makes a set of promises, which can be used by the second goal for its precondition check. If the goal precondition is satisfied based on the promises, the goal can already be formulated (and subsequently selected, expanded, etc.) even if the precondition is not satisfied yet. Additionally, promises are also used as timed initial literals (TILs) for planning in order to expand a goal.

To define promises and how promises affect goal formulation in the context of goal reasoning, we first need to formalize goals. Similar to Cox (2016), we base our definition of a goal on classical planning problems. However, as we need to refer to time later, we assume that each state is timed. Formally, given a finite set of logical atoms A, a state is a pair $(s, t) \in 2^A \times \mathbb{Q}$, where s is the set of logical atoms that are currently true, and t is the current global time. Using the closed world assumption, logical atoms not mentioned in s are implicitly considered to be false. A literal l is either a positive atom (denoted with a) or a negative atom (denoted with $\overline{a}$). For a negative literal l, we also denote the corresponding positive literal with $\overline{l}$ (i.e., $\overline{\overline{l}} = l$). A set of atoms s satisfies a literal l, denoted with $s \models l$ if (1) l is a positive literal a and $a \in s$, or (2) l is a negative literal $\overline{a}$ and $a \notin s$. Given a set of literals $L = \{l_1, \dots, l_n\}$, the state s satisfies L, denoted with $s \models L$, if $s \models l_i$ for each $l_i \in L$.

We define a *goal operator* similar to a planning operator in classical planning (Ghallab, Nau, and Traverso 2016):

Definition 1 (Goal Operator)**.** *A* goal operator *is a tuple* $\gamma = (\text{head}(\gamma), \text{pre}(\gamma), \text{post}(\gamma))$*, where*

- $\text{head}(\gamma)$ *is an expression of the form goal*$(z_1, \dots, z_k)$*, where goal is the* goal name *and* $z_1, \dots, z_k$ *are the* goal parameters*, which include all of the variables that appear in* $\text{pre}(\gamma)$ *and* $\text{post}(\gamma)$*,*
- $\text{pre}(\gamma)$ *is a set of literals describing the condition when a goal may be formulated,*
- $\text{post}(\gamma)$ *is a set of literals describing the objective that the goal pursues, akin to a PDDL goal.*

A goal is a ground instance of a goal operator. A goal g can be *formulated* if $s \models \text{pre}(g)$. When a goal is finished, its post condition holds, i.e., $s \models \text{post}(g)$. Note that in contrast to an action operator (or a macro operator), the effects of a goal are not completely determined; any sequence of actions that satisfies the objective $\text{post}(g)$ is deemed feasible and additional effects are possible. Thus, the action sequence that accomplishes a goal is not pre-determined but computed by a planner and may depend on the current state s. Consider the goal CLEANMACHINE from Listing 1. Its objective defines that the robot carries a container and that the container is filled. However, to achieve this objective given the set preconditions, the robot also moves away from its current location, which will be an additional effect of any plan that achieves the goal.

We have extended the CX with goal operators (shown in Listing 1). For each goal operator and every possible grounding of the operator, the goal reasoner instantiates the corresponding goal precondition and tracks its satisfaction while the system is evolving. Once the goal precondition is satisfied, the goal is formulated. Afterwards, the goal follows the goal lifecycle as before (Figure 1).

5 Promises

A goal reasoning agent relying on goal operators has a model of the conditions that have to be met before it executes a goal g ($\text{pre}(g)$) and a partial model of the world after the successful execution of its plan (its objective $\text{post}(g)$). Thus, when the agent dispatches g, it assumes that it will accomplish the objective at some point in the future (i.e.

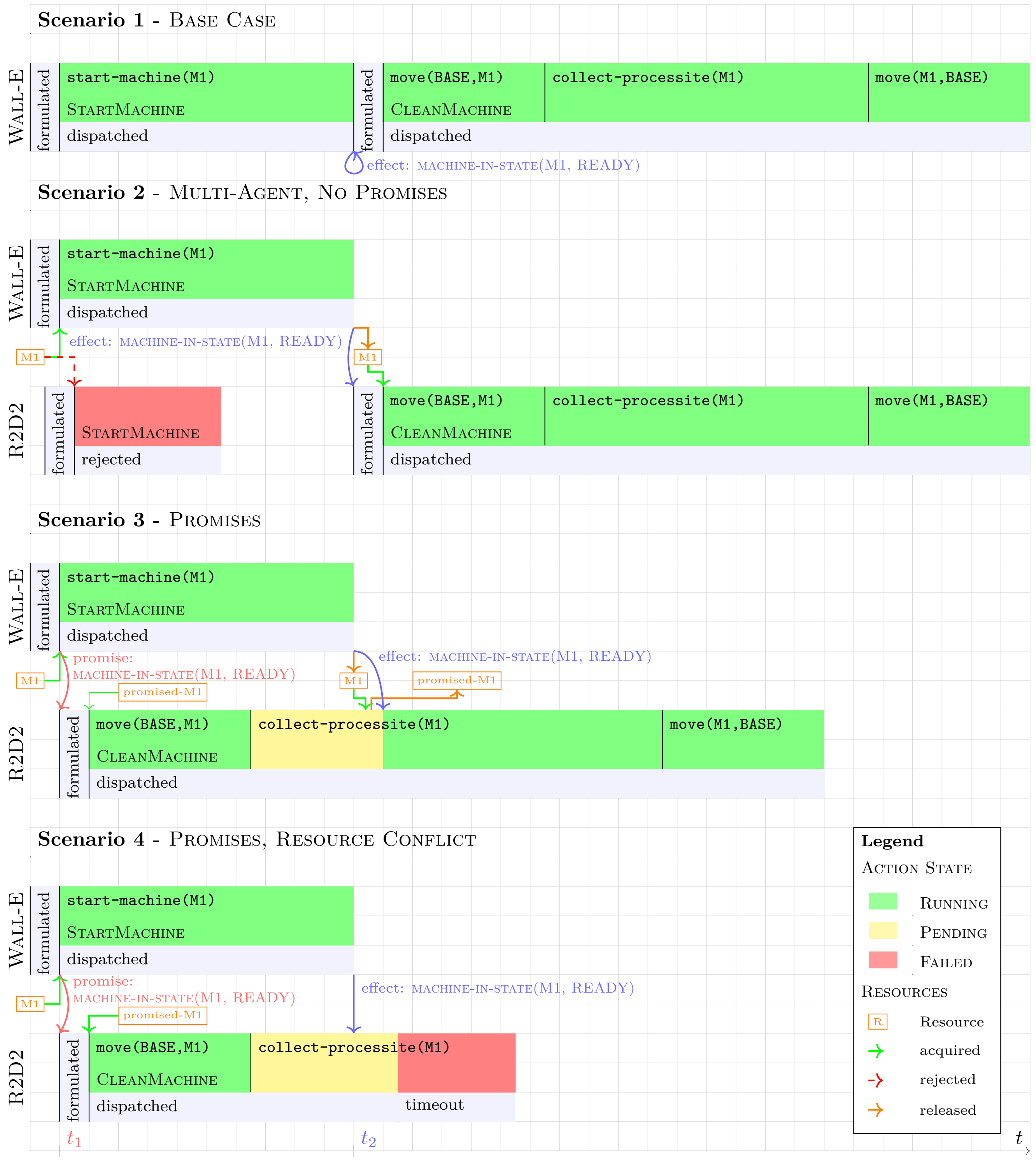


Figure 2: Multiple scenarios of interaction between two goals and robots in the CX with and without promises as discussed in the running example. Scenario 1: one robot Wall-E and no promises. It successively formulates and executes the applicable goals. Scenario 2: two robots (Wall-E and R2D2) and no promises. Both attempt to dispatch StartMachine, illustrating interaction between multiple agents and goal resources. Scenario 3: two robots, promises. R2D2 formulates CleanMachine earlier from the promises, but the action `collect-processite` is pending until the effects from Wall-E are applied. Scenario 4: two robots, promises, and failed resource handover. `collect-processite` is pending until the resource `M1` is handed over. A timeout occurs and the goal fails.

```
(goal-operator (class CleanMachine)
  (param-names     r      side machine  c    mat)
  (param-types     robot loc  machine  cont material)
  (param-quantified)
  (lookahead-time 20)
  (preconditions "
      (and
          (robot-carries ?r ?c)
          (container-can-be-filled ?c)
          (location-is-free ?side)
          (location-is-machine-output ?side)
          (location-part-of-machine ?side ?machine)
          (machine-in-state ?machine READY)
          (machine-makes-material ?machine ?mat)
          (not (storage-is-full))
      )
  ")
  (objective
  "(and (robot-carries ?r ?c)
         (container-filled ?c ?mat)
  )")
)
```

Listing 1: The definition of the goal operator CLEANMACHINE. The precondition defined when a goal may be formulated; the objective defines the PDDL goal to plan for once the goal has been selected.

$s \models \mathrm{post}(g)$). However, the other agents are oblivious to g's objectives and the corresponding changes to the world and therefore need to wait until the first agent has finished its goal. To avoid this unnecessary delay, we introduce *promises*, which intuitively give some guarantee for certain literals to be true in a future state:

Definition 2 (Promise)**.** *A* promise *is a pair (l, t), stating that the literal l will be satisfied at (global) time t.*

Promises are made to the other agents by a *promising* agent when it dispatches a goal. The *receiving* agent may now use those promises to evaluate whether a goal can be formulated, even if its precondition is not satisfied by the world. More precisely, given a set of promises, we can determine the point in time when a goal precondition will be satisfied:

Definition 3 (Promised Time)**.** *Given a state (s, t) and a set of promises $P = \{(l_i, t_i)\}_i$, we define $\mathtt{From}(l, s, t, P)$ as the timepoint when a literal l is satisfied and $\mathtt{Until}(l, s, t, P)$ as the timepoint when l is no longer satisfied:*

$$\mathtt{From}(l, s, t, P) = \begin{cases} t & \textit{if } s \models l \\ \min_{(l,t_i)\in P} t_i & \textit{if } \exists t_i : (l, t_i) \in P \\ \infty & \textit{else} \end{cases}$$

$$\mathtt{Until}(l, s, t, P) = \begin{cases} t & \textit{if } s \models \bar{l} \\ \min_{(\bar{l},t_i)\in P} t_i & \textit{if } \exists t_i : (\bar{l}, t_i) \in P \\ \infty & \textit{else} \end{cases}$$

We extend the definition to a set of literals L:

$$\mathtt{From}(L, s, t, P) = \max_{l_i \in L} \mathtt{From}(l_i, s, t, P)$$
$$\mathtt{Until}(L, s, t, P) = \min_{l_i \in L} \mathtt{Until}(l_i, s, t, P)$$

Promises are tied to goals and can either be based on the effects of the concrete plan actions of an expanded goal, or simply be based on a goal's postcondition. They can be extracted automatically from these definitions, or hand-crafted in simple domains. Though the concept can be applied more flexibly (e.g. dynamic issuing of promises based on active plans and world states), in our implementation promises are static and created during goal expansion.

Goal Formulation with Promises

We continue by describing how promises and promise times for literals can be used for goal formulation. As shown in Listing 1, we augment the goal operator with a *lookahead time*, which is used to evaluate the goal precondition for future points in time. Given a set of promises P, a lookahead time t_l and a goal precondition $\mathrm{pre}(\gamma)$, the goal reasoner formulates a goal in state (s, t) iff the precondition will be satisfied within the next t_l time units, i.e., iff

$$\mathtt{From}(\mathrm{pre}(\gamma), s, t, P) \leq t + t_l$$

Note that if $s \models \mathrm{pre}(\gamma)$, then $\mathtt{From}(\mathrm{pre}(\gamma), s, t, P) = t$ and thus the condition is trivially satisfied. Also, if the lookahead time is set to $t_l = 0$, then the goal is formulated iff its precondition is satisfied in the current state, thus effectively disabling the promise mechanism.

Furthermore, this condition results in an optimistic goal formulation, as the goal will still be formulated even if the goal's precondition is promised to be no longer satisfied at a future point in time. To ensure that a goal is only formulated if its precondition is satisfied for the whole lookahead time, we can additionally require:

$$\mathtt{Until}(\mathrm{pre}(\gamma), s, t, P) \geq t + t_l$$

In our implementation, `Until` is currently not considered, as we are mainly interested in optimistic goal formulation.

Incorporating promises into goal formulation leads to more cooperative behavior between the agents as goals that build on other goals can be considered by an agent during selection, thereby enabling parallelism. Thus, it is a natural extension of the goal lifecycle with an intention sharing mechanism.

Promises in the CX. In the CX, promises are continously evaluated to compute `Until` and `From` for each available grounding of a goal precondition formula. The results are stored for each formula and computed in parallel to the normal formula satisfaction evaluation. Therefore, promises do not directly interfere with the world model, but rather are integrated into the goal formulation process by extending the precondition check to also consider `From` for a certain lookahead time t_l.

The lookahead time is manually chosen based on the expected time to accomplish the objective and are specific for each goal. In scenarios with constant execution times for actions, those can be used directly as the estimate. If execution times for actions are not constant (e.g. driving actions with variable start and end positions), average recorded execution times or more complex estimators might be used, at the loss of promise accuracy. By choosing a lookahead time of 0, promises can be ignored.

Promises are shared between the agents through the shared world model of the CX. For now, we hand-craft promises for each goal operator γ based on $\mathrm{post}(g)$. However, extracting promises automatically from postconditions and plans may be considered in future work. If a goal is completed or fails, the promises are removed from the world model.

Using Promises for Planning

With promises, an agent is able to formulate a goal in expectation of another agent's actions. However, promises also need to be considered when expanding a goal with a PDDL planner. To do so, a promise is translated into a *timed initial literal* (TIL) (Edelkamp and Hoffmann 2004). Similar to a promise, a TIL states that some literal will become true at some point in the future, e.g., `(at 5 (robot-at M1 )` states that `(robot-at M1)` will become true in 5 time steps. We extended the PDDL plan expansion of the CX to translate promises into TILs and to use POPF (Coles et al. 2010), a temporal PDDL planner that supports TILs.

Goal Execution with Promises

When executing a goal that depends on promises (i.e., it has been formulated based on promises), it is necessary to check whether the promise has actually been realized. To do so, we build on top of the execution engine of the CX (Niemueller, Hofmann, and Lakemeyer 2018). For each plan action, the CX continuously checks the action's precondition and only starts executing the action once the precondition is satisfied. While the goal's preconditions can be satisfied (in parts or fully) by relying on promises, they are not applied to action preconditions. Thus, the action execution blocks until the precondition is satisfied. This ensures that actual interactions in the world only operate under the assumption of correct and current world information. Otherwise, actions might fail or lead to unexpected behavior.

Let us revise our examples from SCENARIO 1 and SCENARIO 2 by introducing promises, but ignoring the role of goal resources for now. Similarly to the previous scenarios, WALL-E formulates, selects and executes a goal of class STARTMACHINE on machine `M1`. With promises, the expected outcome of that goal (`machine-in-state(M1, READY)`) will be announced to the other agents when the goal gets dispatched (t_1). R2D2 now uses the provided information to formulate the goal CLEANMACHINE ahead of time, even though the effects of STARTMACHINE have not been applied to the world yet. This behavior is shown in SCENARIO 3 of Figure 2. The first action of the goal's plan (`move`) can be executed directly. However, the precondition of action `collect-processite` is not yet satisifed, since the action `start-machine` is still active on WALL-E until t_2. The executive of R2D2 notices this and `collect-processite` is pending until the effects have been applied. Once the effects are applied, R2D2 can start executing its action. An example of this behavior occuring in simulation is visualized in Figure 3.

Execution Monitoring with Promises

Goals that issue promises might take longer than expected, get stuck, or fail completely, possibly leading to promises that are never realized. Therefore, goals that have been formulated based on promises may have actions whose preconditions will never be satisfied. To deal with this situation, we again rely on execution monitoring. If a pending action is not executable for a certain amount of time, a timeout occurs and the action and its corresponding goal are aborted. However, the timeout spans are increased such that the potentially longer wait times from promise-based execution can be accounted for.

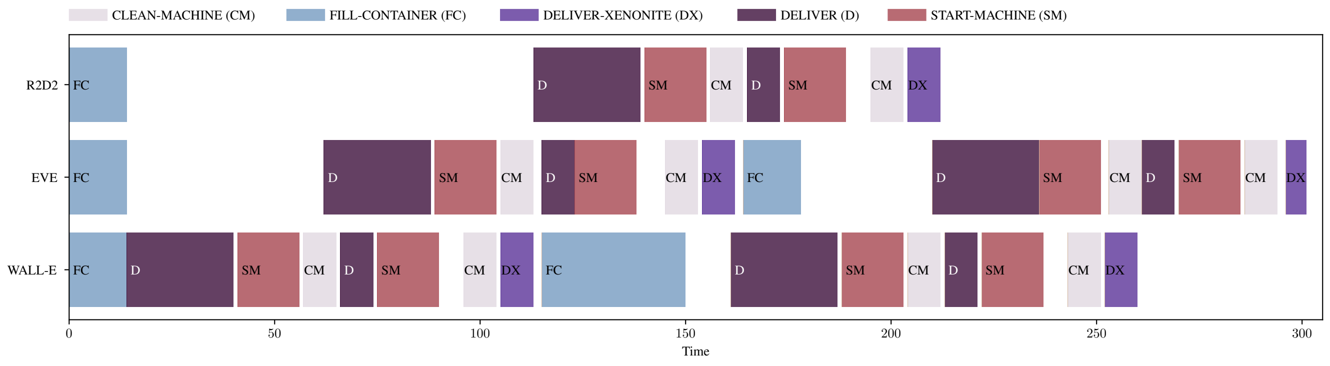
Promises may not cause deadlocks, as a deadlock may only appear if two goals depend on each other w.r.t. promises. However, this is not possible, as promises are used to formulate a goal, but only are issued when the goal is dispatched. Thus, cycles of promises are not possible.

Recall our previous example SCENARIO 3. Action `collect-processite` remains in the state pending until WALL-E finishes the execution of action `start-machine` and its effects are applied. Should WALL-E not complete the action (correctly), execution monitoring will detect the timeout on the side of R2D2 and will trigger new goal formulation. If the promised-from time has elapsed, the promises will not be considered by R2D2 anymore when formulating new goals, leading to a different objective to be chosen by the agent. Should WALL-E manage to fulfill its goal, then the effects will be applied, and R2D2's original goal's preconditions are satisfied through the actual world state.

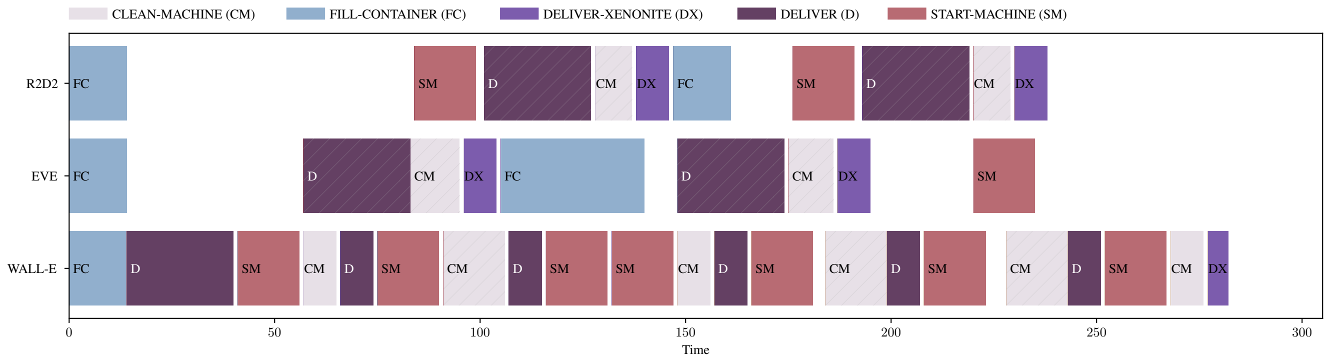
Resource Locks with Promises

In the CX, resource locks at the goal level are used to coordinate conflicting goals. Naturally, goals that are formulated on promises often rely on some of the same resources that are required by the goal that issued the promise. A goal that fills a machine requires the machine as a resource. Another goal that is promise-dependent and operates on the same machine might require it as a resoure as well. In this scenario, the second goal may be formulated but would immediately be rejected, as the required resource is still held by the promising goal. To resolve this issue, promise-dependent goals will delay the resource acquisition for any resource that is currently held by the promising goal. As soon as the promising goal is finished and therefore the resource is released, the resource is acquired by the promise-dependent goal. To make sure that there is no conflict between two different promise-dependent goals, for each such delayed resource, a *promise resource* is acquired, which is simply the main resource prefixed with `promised-`. Effectively, a promise-dependent goal first acquires the promise resource, then, as soon as the promising goal releases the resource, the promise-dependent goal acquires the main resource, and then releases the promise resource.

To illustrate the mechanism, let us once again consider the example from SCENARIO 3. When dispatched, the goal STARTMACHINE holds the resource `M1` and R2D2 formulates the goal CLEANMACHINE based on WALL-E's promise. Since the goal operates on the same machine, it needs to acquire the same goal resource. As the



(a) A run without promises. Each goal formulation must wait until the previous goal has finished.



(b) A run with promises. The striped goals have been formulated based on promises. Compared to the run without promises, the promise-dependent goals (e.g., the first CLEANMACHINE goal by EVE) are dispatched earlier, because a future effect of another goal (i.e. the second STARTMACHINE goal by WALL-E) is considered for goal formulation and expansion, thus leading to an overall better performance.

Figure 3: A comparison of two exemplary runs in the Xenonite domain.

goal is promise-dependent, it first acquires the resource `promised-M1`. This resource is currently held by no other agent, thus it can be acquired and the goal dispatched. R2D2 first executes the action `move(BASE, M1)`. As before, the next action `collect-processite(M1)` remains pending until WALL-E has completed its goal. At this point, WALL-E releases the resource `M1`, which is then acquired by R2D2. Should this resource handover fail, e.g., because the goal STARTMACHINE by WALL-E never releases its resources, the action eventually times out and R2D2's goal is aborted, as shown in SCENARIO 4 of Figure 2.

6 Evaluation

We evaluate our prototypical implementation[1] of promises for multi-agent cooperation as an extension of the CX in the same simplified production logistics scenario that we used as a running example throughout this paper. In our evaluation scenario, three robots were tasked with filling $n = 5$ containers with the materials Xenonite and to deliver it to a storage area. The number of containers was intentionally chosen to be larger than the number of robots to simulate an oversubscription problem and highlight the effects of better cooperation on task efficiency. The three robots first collect raw materials and then refine them stepwise by passing them through a series of two machines in sequence. Once all containers are filled and stored, the objective is completed.

We compare the performance of the three robots using POPF to expand goals. Figure 2 shows how promises can lead to faster execution of goals by starting a goal (which always starts with a move action) while the goal's precondition (e.g., that the machine is in a certain state) is not fulfilled yet. Figure 3 shows exemplary runs of three robots producing and delivering 5 containers of Xenonite. The expected behavior as highlighted in Figure 2 can indeed be observed in the simulation runs.

To compare the performance, we simulated each scenario five times. All experiments were carried out on an Intel Core i7 1165G7 machine with 32 GB of RAM. Without promises, the three robots needed $303\,\mathrm{sec}$ to fill and store all containers, compared to $(284.40 \pm 0.55)\,\mathrm{sec}$ with promises. At each call, POPF needed less than $1\,\mathrm{sec}$ to generate a plan for the given goal. This shows, at least in this simplified scenario, that promises lead to more effective collaboration, and that the planner can make use of promises when expanding a goal.

[1]https://doi.org/10.5281/zenodo.6610426

7 Conclusion

In multi-agent goal reasoning scenarios, effective collaboration is often difficult, as one agent is unaware of the goals pursued by the other agents. We have introduced *promises*, a method for intention sharing in a goal reasoning framework. When dispatching a goal, each agent promises a set of literals that will be true at some future timepoint, which can be used by another agent to formulate and dispatch goals early. We have described how promises can be defined based on *goal operators*, which describe goals similar to how action operators describe action instances. By translating promises to timed initial literals, we allow the PDDL planner to make use of promises during goal expansion. The evaluation showed that using promises with our prototypical implementation improved the performance of a team of three robots in a simplified logistics scenario. For future work, we plan to use and evaluate promises in a real-world scenario from the RoboCup Logistics League (RCLL) (Niemueller et al. 2016) and compare it against our distributed multi-agent goal reasoning system (Hofmann et al. 2021).

Acknowledgements

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 2236/1, the EU ICT-48 2020 project TAILOR (No. 952215), and Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612.

References

Aha, D. W. 2018. Goal Reasoning: Foundations, Emerging Applications, and Prospects. *AI Magazine*, 39(2): 3–24.

Albrecht, S. V.; and Stone, P. 2018. Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems. *Artificial Intelligence*, 258: 66–95.

Brenner, M.; and Nebel, B. 2009. Continual Planning and Acting in Dynamic Multiagent Environments. *Autonomous Agents and Multi-Agent Systems*, 19(3).

Carberry, S. 2001. Techniques for Plan Recognition. *User Modeling and User-Adapted Interaction*, 11(1): 31–48.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proceedings of the 20th International Conference on International Conference on Automated Planning and Scheduling (ICAPS)*, ICAPS'10, 42–49. AAAI Press.

Cox, J. S.; and Durfee, E. H. 2005. An Efficient Algorithm for Multiagent Plan Coordination. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, 828–835. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-59593-093-4.

Cox, J. S.; Durfee, E. H.; and Bartold, T. 2005. A Distributed Framework for Solving the Multiagent Plan Coordination Problem. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, 821–827. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-59593-093-4.

Cox, M. T. 2016. A Model of Planning, Action, and Interpretation with Goal Reasoning. In *Proceedings of the 4th Annual Conference on Advances in Cognitive Systems*, 48–63. Palo Alto, CA; USA: Cognitive Systems Foundation.

Davis, R.; and Smith, R. G. 1983. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence*, 20(1): 63–109.

Dias, M.; Zlot, R.; Kalra, N.; and Stentz, A. 2006. Market-Based Multirobot Coordination: A Survey and Analysis. *Proceedings of the IEEE*, 94(7): 1257–1270.

Dignum, F.; Dunin-Keplicz, B.; and Verbrugge, R. 2001. Agent Theory for Team Formation by Dialogue. In Castelfranchi, C.; and Lespérance, Y., eds., *Intelligent Agents VII Agent Theories Architectures and Languages*, Lecture Notes in Computer Science, 150–166. Berlin, Heidelberg: Springer. ISBN 978-3-540-44631-6.

Edelkamp, S.; and Hoffmann, J. 2004. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical report, Technical Report 195, University of Freiburg.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press. ISBN 978-1-139-58392-3.

Grant, J.; Kraus, S.; and Perlis, D. 2002. A Logic-Based Model of Intentions for Multi-Agent Subcontracting. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 320–325.

Grosz, B. J.; and Kraus, S. 1996. Collaborative Plans for Complex Group Action. *Artificial Intelligence*, 86(2): 269–357.

Hertle, A.; and Nebel, B. 2018. Efficient Auction Based Coordination for Distributed Multi-agent Planning in Temporal Domains Using Resource Abstraction. In Trollmann, F.; and Turhan, A.-Y., eds., *KI 2018: Advances in Artificial Intelligence*, volume 11117, 86–98. Cham: Springer International Publishing. ISBN 978-3-030-00110-0 978-3-030-00111-7.

Hoffmann, J.; and Brafman, R. I. 2005. Contingent Planning via Heuristic Forward Search with Implicit Belief States. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 71–80. ISBN 1-57735-220-3.

Hoffmann, J.; and Brafman, R. I. 2006. Conformant Planning via Heuristic Forward Search: A New Approach. *Artificial Intelligence*, 170(6-7).

Hofmann, T.; Niemueller, T.; Claßen, J.; and Lakemeyer, G. 2016. Continual Planning in Golog. In *Proceedings of the 30th Conference on Artificial Intelligence (AAAI)*. Phoenix, AZ; USA.

Hofmann, T.; Viehmann, T.; Gomaa, M.; Habering, D.; Niemueller, T.; and Lakemeyer, G. 2021. Multi-Agent Goal Reasoning with the CLIPS Executive in the RoboCup Logistics League. In *Proceedings of the 13th International Conference on Agents and Artifical Intelligence (ICAART)*.

Holvoet, T.; and Valckenaers, P. 2006. Beliefs, Desires and Intentions through the Environment. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS '06*, 1052. Hakodate, Japan: ACM Press. ISBN 978-1-59593-303-4.

Horling, B.; and Lesser, V. 2004. A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, 19(4): 281–316.

Iocchi, L.; Nardi, D.; Piaggio, M.; and Sgorbissa, A. 2003. Distributed Coordination in Heterogeneous Multi-Robot Systems. *Autonomous Robots*, 15(2): 155–168.

Jin, L.; Li, S.; La, H. M.; Zhang, X.; and Hu, B. 2019. Dynamic Task Allocation in Multi-Robot Coordination for Moving Target Tracking: A Distributed Approach. *Automatica*, 100: 75–81.

Jonsson, A.; and Rovatsos, M. 2011. Scaling Up Multiagent Planning: A Best-Response Approach. In *Twenty-First International Conference on Automated Planning and Scheduling*.

Kaelbling, L. P.; and Lozano-Pérez, T. 2011. Hierarchical Task and Motion Planning in the Now. In *2011 IEEE International Conference on Robotics and Automation*, 1470–1477.

Kraus, S. 2001. Automated Negotiation and Decision Making in Multiagent Environments. In Luck, M.; Mařík, V.; Štěpánková, O.; and Trappl, R., eds., *Multi-Agent Systems and Applications: 9th ECCAI Advanced Course, ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, EASSS 2001 Prague, Czech Republic, July 2–13, 2001 Selected Tutorial Papers*, Lecture Notes in Computer Science, 150–172. Berlin, Heidelberg: Springer. ISBN 978-3-540-47745-7.

Kraus, S.; Sycara, K.; and Evenchik, A. 1998. Reaching Agreements through Argumentation: A Logical Model and Implementation. *Artificial Intelligence*, 104(1): 1–69.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A Deliberative Architecture for AUV Control. In *2008 IEEE International Conference on Robotics and Automation*, 1049–1054.

Muñoz-Avila, H.; Jaidee, U.; Aha, D. W.; and Carter, E. 2010. Goal-Driven Autonomy with Case-Based Reasoning. In Bichindaritz, I.; and Montani, S., eds., *Case-Based Reasoning. Research and Development*, Lecture Notes in Computer Science, 228–241. Berlin, Heidelberg: Springer. ISBN 978-3-642-14274-1.

Niemueller, T.; Hofmann, T.; and Lakemeyer, G. 2018. CLIPS-based Execution for PDDL Planners. In *2nd ICAPS Workshop on Integrated Planning, Acting, and Execution (IntEx)*. Delft, Netherlands.

Niemueller, T.; Hofmann, T.; and Lakemeyer, G. 2019. Goal Reasoning in the CLIPS Executive for Integrated Planning and Execution. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*, 1, 754–763. Berkeley, CA, USA.

Niemueller, T.; Karpas, E.; Vaquero, T.; and Timmons, E. 2016. Planning Competition for Logistics Robots in Simulation. In *4th ICAPS Workshop on Planning and Robotics (PlanRob)*.

Parker, L. 1998. ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation. *IEEE Transactions on Robotics and Automation*, 14(2): 220–240.

Pinyol, I.; and Sabater-Mir, J. 2013. Computational Trust and Reputation Models for Open Multi-Agent Systems: A Review. *Artificial Intelligence Review*, 40(1): 1–25.

Roberts, M.; Apker, T.; Johnson, B.; Auslander, B.; Wellman, B.; and Aha, D. W. 2015. Coordinating Robot Teams for Disaster Relief. In *Proceedings of the 28th Florida Artificial Intelligence Research Society Conference*. Hollywood, FL: AAAI Press.

Roberts, M.; Hiatt, L.; Coman, A.; Choi, D.; Johnson, B.; and Aha, D. 2016. ActorSim, a Toolkit for Studying Cross-Disciplinary Challenges in Autonomy. In *AAAI Fall Symposium on Cross-Disciplinary Challenges for Autonomous Systems*.

Roberts, M.; Hiatt, L. M.; Shetty, V.; Brumback, B.; Enochs, B.; and Jampathom, P. 2021. Goal Lifecycle Networks for Robotics. In *The International FLAIRS Conference Proceedings*, volume 34.

Roberts, M.; Vattam, S.; Alford, R.; Auslander, B.; Karneeb, J.; Molineaux, M.; Apker, T.; Wilson, M.; McMahon, J.; and Aha, D. W. 2014. Iterative Goal Refinement for Robotics. In *1st ICAPS Workshop on Planning in Robotics (PlanRob)*.

Sarratt, T.; and Jhala, A. 2016. Policy Communication for Coordination with Unknown Teammates. In *The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence*, 567–573.

Shivashankar, V.; Alford, R.; Kuter, U.; and Nau, D. 2013. The GoDeL Planning System: A More Perfect Union of Domain-Independent and Hierarchical Planning. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, 2380–2386. Beijing, China: AAAI Press. ISBN 978-1-57735-633-2.

Shivashankar, V.; Kuter, U.; Nau, D.; and Alford, R. 2012. A Hierarchical Goal-based Formalism and Algorithm for Single-agent Planning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '12, 981–988. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-2-3.

Sukthankar, G.; Geib, C.; Bui, H.; Pynadath, D.; and Goldman, R. P. 2014. *Plan, Activity, and Intent Recognition: Theory and Practice*. Newnes. ISBN 978-0-12-401710-8.

Vail, D.; and Veloso, M. 2003. Dynamic Multi-Robot Coordination. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume 2, 87–100.

Vattam, S.; Klenk, M.; Molineaux, M.; and Aha, D. W. 2013. Breadth of Approaches to Goal Reasoning: A Research Survey. In *1st CogSys Workshop on Goal Reasoning*.

Verma, J. K.; and Ranga, V. 2021. Multi-Robot Coordination Analysis, Taxonomy, Challenges and Future Scope. *Journal of Intelligent & Robotic Systems*, 102(1): 10.

Wilson, M.; and Aha, D. W. 2021. A Goal Reasoning Model for Autonomous Underwater Vehicles. In *Proceedings of the Ninth Goal Reasoning Workshop*. Online.

Wygant, R. M. 1989. CLIPS — A Powerful Development and Delivery Expert System Tool. *Computers & Industrial Engineering*, 17(1-4): 546–549.

Yu, H.; Shen, Z.; Leung, C.; Miao, C.; and Lesser, V. R. 2013. A Survey of Multi-Agent Trust Management Systems. *IEEE Access*, 1: 35–50.